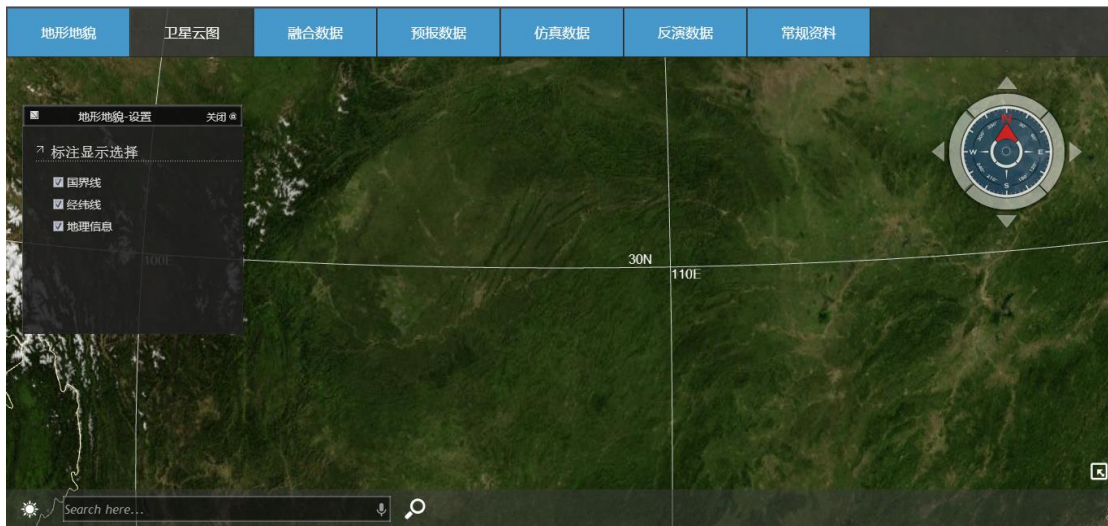


# 本周周报（11.25 -12.1）

刘昊南

## 本周工作

1. 完成了金字塔计算模块，能够实时地计算当前的地球纹理层次以及可见的区块，在此基础上实现了纹理的 lod



2. 这次实现的金字塔计算与 pc 端的实现方法有了很大的提高。用我的笔记本电脑跑客户端程序的时候，当进入较精细的层次后，由于区块更小，面片更多，程序相当卡，计算可见块的效率较低。如果 web 端照套 pc 端方法的话，根本跑不动。构思了很久，对 web 端的计算方法，目前即使在最精细的层次，可见块的计算也能做到实时的，十分流畅。总结起来有以下几点优化：

- a) 改进了层次计算的效率：之前在 pc 端，层次是这样计算的：当前相机高度→地球上的可见经纬度范围→屏幕上每个像素所代表的经纬度→当前层次。这样每一帧都要经过较多步骤的三角函数计算来决定当前层次，相当繁琐。

Web 端是这样计算的，首先对上述步骤进行反推，即：当前层次→屏幕屏幕上每个像素所代表的经纬度→地球上的可见经纬度范围→当前相机高度。这样我们就可以得到每个层次的边界高度，在程序加载时，我们把第 1 层到第 n 层的边界高度计算出来，缓存到一个 table 里。然后每一帧的层次计算变为：当前相机高度→查表得到当前层次。减少了每一帧的计算量。

- b) 改进了可见块的计算效率：之前在 pc 端没有采用任何的优化方法，对于每一个块，判断是否可见的条件是，一个块的所有三角面片的所有顶点中，有一个顶点在视景体内。这导致了在较精细的层次，块的数目呈指数级增长，三角面片更是不计其数，计算量十分大，cpu 占用过多，导致程序卡顿。在台式机上，由于 cpu 性能强劲，

表现不太明显，但是在笔记本上影响较大。

在 Web 端进行了如下优化：相机位置→地球上能被相机位置点光源照亮的经纬度范围→能被相机位置点光源照亮的块→根据块的四个角的顶点构建包围盒→判断包围盒是否与视景体相交→可见块。

地球上能被相机位置点光源照亮的经纬度范围：进行粗略的筛选，即相机的 fov 为 180 度时的地球的可见经纬度范围。当层次越来越精细，经纬度范围也越来越小，筛选出来块的数目会出现先增加后减小的趋势，即筛选出的块数目较少，同时地球背面的块即使在视景体之内也能被剔除掉。

根据块的四个角的顶点构建包围盒：根据球面上块的几何特点，根据块的四个角的顶点就能构建出它的包围盒，这样就不用考虑块内部的大量三角面片，减少了大量计算。

判断包围盒是否与视景体相交：为此专门实现了一个 `frustum` 类，用于视景体剔除，实现了对 `point`、`boundingbox` 和 `boundingsphere` 的高效剔除方法。在这里对粗略筛选出来的块进行精确筛选，能够快速得到可见块。

## 下周计划

1. 实现在 `webgl` 中的 `ray casting`，设计体绘制框架